Developers guide to `phylobase`

Peter Cowan, Ben Bolker & other developers of `phylobase`

November 2, 2009

Contents

1 Introduction

This vignette is intended as a guide for the development of `phylobase` and a repository of technical notes of primary interest to it's developers and others interested in gory details — by contrast the other `phylobase` vignette is more of an introduction & user's manual.

2 R-Forge

Our development infrastructure hosted by R-Forge,[1] and includes a Subversion repository (see Appendix A [Subversion]) for source code management, trackers for bug reports and feature requests,[2] and mailing lists for development discussion and tracking changes in the source code.[3]

The R-Forge manual[4] describes how to develop a package using their infrastructure Of particular importance is the section on getting ssh keys to work so that you can commit changes to the `phylobase` code using Subversion (`svn`).

On feature of R-Forge is a package repository that allows prerelease versions of `phylobase` to be install directly from `R`.

```
> install.packages("phylobase", repos = "http://r-forge.r-project.org")
```

3 Building `phylobase`

If you are interested in building `phylobase` from source, you will need the same tools required to build `R` itself. These are documented in the R Installation and Administration manual.[5] If you are running Mac OS X you'll need to have the developer tools.[6] On Windows you'll need to install the "Windows toolset"[7] as described in the above manual. If you are running Linux, you probably already know what you need. To build the vignettes and LaTeX documentation you will need to install a TeX distribution like MacTex[8] for Mac OS X, Tex-Live for Linux, or MiKTeX[9] for Windows. Once you have the proper tools installed follow the instruction in the Writing `R` Extensions manual.[10]

4 Coding standards

We try to follow the coding standards of Bioconductor.[11] Namely, variables and particularily function exposed to users should be in `camelCase`. As of this writing many non-exported functions are prefixed with '.' however, future private functions should be named without this convention. Indentation should be 4 spaces and tabs should be avoided. We also use `<-` for assignment, and place spaces after commas, in indexes and function calls (e.g. `df[2, 2]` and `seq(1, 2, 0.1)`, not `df[2,2]` or `seq(1,2,0.1`.)

[1] http://phylobase.r-forge.r-project.org
[2] http://r-forge.r-project.org/tracker/?group_id=111
[3] http://r-forge.r-project.org/mail/?group_id=111
[4] http://r-forge.r-project.org/R-Forge_Manual.pdf
[5] http://cran.r-project.org/doc/manuals/R-admin.html
[6] http://developer.apple.com/technology/xcode.html
[7] http://www.murdoch-sutherland.com/Rtools/
[8] http://www.tug.org/mactex/
[9] http://www.miktex.org/
[10] http://cran.r-project.org/doc/manuals/R-exts.html
[11] http://wiki.fhcrc.org/bioc/Coding_Standards

5 Release procedure

When the package code has stabilized or significant bugs have been fixed we want to push those changes out to CRAN. This section documents the release sequence used for when submitting the package to CRAN.

1. Update the NEWS file which resides in the pkg/ directory and describes changes in the package since its last release. New changes should be appended above the earlier entries and formatted so that it can be read by `news(package="phylobase")`, (format documented in `?news`). Briefly, the version header should look like a previous entry and changes should be grouped in categories which start at the beginning of a line. Within each category individual changes should be marked with an indented (4 spaces) asterisk, with the change text indented and wrapped a further four space, eight total. The most comprehensive way to find the changes since the last version is to look at the SVN log. This procedure was followed for the the 0.5 release and assumes that you have the entire project checked out, including the www and tags directories.

   Navigate to the tag for the previous release and get the revision it was created:

   ```
   ~$ cd phylobase/tags/phylobase-0.4
   ~$ svn log --stop-on-copy
   ------------------------------------------------------------------------
   r309 | skembel | 2008-12-18 12:55:14 -0800 (Thu, 18 Dec 2008) | 1 line

   Tagging current version as 0.4 prior to hackathon changes
   ------------------------------------------------------------------------
   ```

   In this case we can see that the tag was created in revision 309 and as this was the current release, we want all the changes that have been made since. To do that we navigate to the top level to get the change log from any branches that have been made and save the log. If there are unmerged branches, care should be taken to exclude those changes from the NEWS file. Because this file may be rather large we'll output it to a file.

   ```
   ~$ cd ../..
   ~$ svn log -r309:HEAD > RecentChanges.txt
   ```

   The NEWS file can then be updated by going through the RecentChanges.txt file and picking out significant changes.

2. The DESCRIPTION file should be updated to reflect the new version number and current date. Version numbers should follow the 0.5.0 format (no dashes), and the date should be formatted as 2009-01-30.

3. Rebuild the vignettes to incorporate the latest changes. CRAN may not have all the package we use to build our vignettes or able to run latex the multiple time necessary to generate the PDFs.

   ```
   ~$ cd phylobase/pkg/inst/doc
   ~$ R CMD Sweave phylobase.Rnw
   ~$ pdflatex phylobase.tex
   ```

```
~$ pdflatex phylobase.tex

~$ R CMD Sweave developer.Rnw
~$ pdflatex developer.tex
~$ pdflatex developer.tex

~$ rm rm *.toc *.out *.log *.aux
```

4. Code freeze. Before the package can be submitted to CRAN it must pass the R-Forge build and check process which happens every night. The easiest way to handle this is to freeze the the code for a day or two after the NEWS, DESCRIPTION and PDF files have been updated.

5. Tag the release in SVN. Each release is tagged so that a copy of it is easily available if needed at a later date. This is done using the SVN copy as follows for a hypothetical 0.5.1 release:

```
~$ cd phylobase/
~$ svn copy pkg/ tags/phylobase-0.5.1
~$ svn commit -m "tagging version 0.5.1"
```

6. Uploading to CRAN is done by clicking the Upload to CRAN link on the R-Forge package page. Ensure that the revision number corresponds to the revision with the updated NEWS etc. files.

7. Update the R-Forge website with the correct version number and PDFs of the vignettes.

6 Unit testing with RUnit

We are in the process of moving our testing infrastructure to the `RUnit`[12] framework. New contributions and bug fixes should be accompanied by unit tests which test the basic functionality of the code as well as edge cases (e.g what happens when the function is passed an empty string or negative number – even when those inputs don't make sense.) Unit tests are stored in the `inst/unitTests/` directory and are named according to the source file they correspond to. See the included tests and the `RUnit` documentation for further details. `RUnit` has a few advantages over the other testing frameworks in `R`, namely the examples in documentation, vignettes,

7 Non-exported functions

These functions are for internal use in `phylobase` not exported. Since most are not documented elsewhere, they are documented here.

`.chnumsort` A convenience function that coerces vector of strings to numbers for sorting, then coerces the vector back to stings. Currently only used inside the `prune` method.

`.createEdge`

[12] http://cran.r-project.org/web/packages/RUnit/index.html

`.createLabels` Used any time labels are needed, including when updating the labels via `labels()<-` or constructing a `phylo4` object. It takes a vector of names to use or `NULL` if new labels should be generated, integers indicating the number of tips and internal nodes, as well as a string to indicated the type of labels to generate "tip" and "internal", for either tip or internal labels along or "allnode" or all nodes.

`.genlab` A handy function that can generate labels for applying to nodes and tips. The function takes to arguments, a 'base' string, and an integer indicating the number of labels desired. The result is a vector of string with a number (padded with '0') suffix e.g. `foo01...foo12`. This function is used to generate names in the `.createLabels` function as well as for for making temporary names during the `prune` method.

`.phylo4Data`

`.phylo4ToDataFrame`

`.bubLegendGrob` This function generates a `Grid` graphics object (a grob) for drawing the `phylobubbles()` legend. For reason I have not been able to understand, it must be defined outside of the `phylobubbles` function. It takes the raw tip data values for a `phylo4d` object as well as the scaled values used for making the bubble plot, both of these vectors are passed directly to the `drawDetails.bubLegend` function.

`drawDetails.bubLegend` This function is the `drawDetails` method for the `bubLegend` grob described above. The `drawDetails` method is called every time a plot is generated or resized. In this case it calculates labels and sizes for drawing the example bubbles in the phylobubbles legend. This is necessary because the because the main bubble plot bubble can change in size as the plot is resized. Because the legend and the bubble plot occur in different viewports the legend cannot know the size of the main plot circles (they are plotted relative to the space available in their viewport.)

`orderIndex` This function is called from the `reorder` method. It takes a `phylo4` or `phylo4d` object and a string indicating the desired tree ordering, currently one of "preorder" or "postorder". It's value is a vector indicating the respective ordering of the edge matrix from top to bottom (i.e. in postorder the first edge in the edge matrix would terminate in a tip, while in preorder the first edge would be the root edge.

8 The Nexus Class Library and Rcpp

9 S4 classes and methods

10 Grid graphics

11 Converting between tree formats and "round-trip" issues

We should in principle be able to convert from other formats to `phylo4(d)` and (`ape::phylo`, `ouch::ouch` etc.) and back without losing any information. The two classes of exceptions would be (1) where `phylo4` stores *less* information than the other formats (we would try to avoid this), and (2) where there are ambiguities etc. in the other formats (we would try to avoid this, too,

but it may be difficult; ideally we would consult the package maintainers and try to get them to eliminate the ambiguities in their formats).

Ideally we would be able to use `identical()` to test equality — this tests "bit-by-bit" equality, and is intolerant of *any* differences in format. More loosely, `all.equal()` allows for numeric variation below a certain tolerance, etc. (these correspond to `RUnit::checkEquals()` and `RUnit::checkIdentical()`).

Case in point: `ape` is not entirely consistent in its internal representations, which causes some difficulty in creating perfect round trips (see `tests/roundtrip.R` for workarounds). In particular,

- `unroot()` contains several statements that subtract 1 from components of the data structure that were previously stored as `integer`. Because 1 is subtracted an not `1L` (an explicitly integer constant), this coerces those elements to be `numeric` instead.
- different ways of creating trees in `ape` (`read.tree()`, `rcoal()`) generate structures with the internal elements in different orders. When `phylobase` re-exports them, it always uses the order {`edge`, `edge.length`, `tip.label`, `Nnode`, [`node.label`], [`root.edge`]}, which matches the trees produced by `rcoal` but not those produced by `read.tree`
- because of differences in ordering standards, it's not clear that we can always preserve ordering information through non-trivial manipulations in phylobase

A Subversion

A.1 Organization of the `phylobase` repository

The `phylobase` subversion repository is organized with four top level directories

- `branches/` where changes that will result in large disruption of the package take place
- `pkg/` where the main trunk of the project exists. This directory is built and available through the R-Forge install mechanism so it is important that it passes `R CMD check`
- `tags/` where copies of previous `phylobase` releases are kept
- `www/` where the R-Forge webpage for phylobase is maintained. This must be kept and the top level for R-Forge to find it

A.2 Using subversion for the first time

The general a subversion work flow goes as follows. I've written this with macs linux in mind, however windows users will follow the same work flow but probably use a graphical front end for subversion, like TortoiseSVN.[13]

R-Forge allows code to be checked out in two different ways. First, you can get the code anonymously using this terminal command:

```
svn checkout \
svn://svn.r-forge.r-project.org/svnroot/phylobase
```

[13] http://tortoisesvn.tigris.org/

If you would like to make commits directly to the source repository you need to register with R-Forge and be added to the `phylobase` project as a developer. Once you've done this, you'll need to have the ssh key set up in your R-Forge account. These terminal commands will make a copy of the folder "phylobase", and all the source files for the package, in whatever directory you are in (in this case " `/Code`"), change "`[name]`" to your R-Forge username.

```
cd ~/Code/
svn checkout \
svn+ssh://[name]@svn.r-forge.r-project.org/svnroot/phylobase
```

Once you have a copy of the package, hack away at it and adding functions and documentation. Save changes. Then check to make sure you have the latest version of the package, it is often the case that another developer has committed a change while you were working.

```
svn update
```

R provides some tools for checking packages. They help ensure that the package can be installed and that all the proper documentation has been added. To keep the repository clean of files that are created during the build process copy the package folder to a tmp folder before running the check.

```
cp -R ~/Code/phylobase/pkg/ ~/Code/phylobase-tmp/
R CMD check ~/Code/phylobase-tmp
```

Fix any errors or warnings that come up, and repeat `R CMD check` as necessary. The R core developers provide a manual [14] for writing R Extensions (packages) which describes the package and documentation formats. Main gotchas are being sure that you've properly updated the `DESCRIPTION` and `NAMESPACE` files and ensuring that the documentation is in the proper format. `R CMD check` warnings/errors are very useful for helping figure out what the issue is. And, the command `prompt()` will provide "fill-in-the-blank" documentation if you're documenting a function for the first time.

The next step is to take a look at what we've actually changed. The `status` command will show any file that's been added (`A`), modified (`M`), or is unknown (and may need to be added) to subversion (`?`).

```
svn status
```

For instance if I've added a new file called foo.R subversion doesn't follow it until I tell it to. So I might see something like:

```
? fooBar.R
```

Which we can remedy with

```
svn add fooBar.R
```

svn status will then show:

```
A fooBar.R
```

[14] http://cran.r-project.org/doc/manuals/R-exts.pdf

Update again for good measure (you can't overdo it on this).

```
svn up
```

And, finally commit the changes with a helpful message (the -m portion) about what they change does:

```
svn commit fooBar.R -m "Function fooBar for calculating foo on the class bar"
```

Subversion, has a whole bag of tricks, full documentation of which can be found in the subversion book.[15] There are also a number of graphical interface programs that you can use with Subversion as well.

A.3 Subversion patches

If you don't have developer access to the project but you'd like to fix a bug or add a feature you can provide the change as a patch. To make a patch with subversion first get a copy of the most recent code.

```
svn checkout svn://svn.r-forge.r-project.org/svnroot/phylobase/pkg
```

Make whatever changes in the code or documentation are necessary. For instance if there is a simple function `helloWorld`, in a file called fooBar.R

```
helloWorld <- function(){
    # This is a comment
    print('Hello World!')
}
```

And, I want to update it, would find the file fooBar.R in my svn checkout and change it to:

```
helloWorld <- function(x){
    # This is a comment
    # This is a new comment
    print(paste('Hello', x, '!')
}
```

To supply that change to a developer with subversion access I would make a diff file of the change. The way to make a diff is to run the `svn diff` command on the file in question.

```
svn diff fooBar.R > helloWorld.diff
```

This however just spits the output of `svn diff` to the terminal. To save the output to a file use ">" which tells the terminal to save the output to a file, this file should end in ".diff" and will have pluses and minuses to indicate which lines have been added and removed.

```
svn diff fooBar.R > fooBar.diff
```

[15] http://svnbook.red-bean.com/

```
--- /Users/birch/fooBar.R
+++ /Users/birch/fooBar.R
-helloWorld <- function(){
+helloWorld <- function(x){
     # This is a comment
-    print('Hello World!')
+    # This is a new comment
+    print(paste('Hello', x, '!')
 }
```

The fooBar.diff file can then be sent to the developers mailing list or the R-Forge Patches issue tracker.[16] Where another developer can easily review and apply the patch.

A.4 Branching and merging with svn

[16] https://r-forge.r-project.org/tracker/?atid=490&group_id=111&func=browse